Finding Fault with Faults: A Case Study

Allen P. Nikora
Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91109-8099
Mail Stop 264-805
vox: (818)393-1104
fax: (818)393-7830
Allen. P. Nikora@jpl.nasa.gov

John C. Munson
Computer Science Department
University ofIdaho
Moscow, ID 83844-1010
vox: (208)885-7789
fax: (208)8885-9052
jmunson@cs.uidaho. edu

ABSTRACT

Over the past several years, significant effort has been devoted to the process of predicting software system fault content during the earlier development phases. Much of this work has involved relating structural characteristics of software systems (e.g. complexity measurements of the source code and design) to the number of faults in the system. We describe our effort in extending this work beyond the initial software construction. Our area offocus is determining the rate offault injection over a sequence of successive builds, first observing that software faults may be seen to fall into two distinct classes - some faults are incorporated during the initial coding effort, while others are added in successive software builds. Experience in working with NASA software development efforts is discussed, including practical issues in obtaining data and assuring its validity. One of the most significant topics discussed is the methodology for the precise determination of a fault condition and the mapping of software faults to individual program modules. We examine the results obtained to date, and conclude with a description of our plans to extend this work in the future.

FINDING FAULT WITH FAULTS· A CASE STUDY

Allen P. Nikora
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA
Allen.P.Nikora@jpl.nasa.gov

John C. Munson
Computer Science Department
University of Idaho
Moscow, ID
jmunson@cs.uidaho.edu

Annual Oregon Worksh⊃≥ on Software Metrics
May 11-13, 1997
Coeur d'Alene, ID

Overview

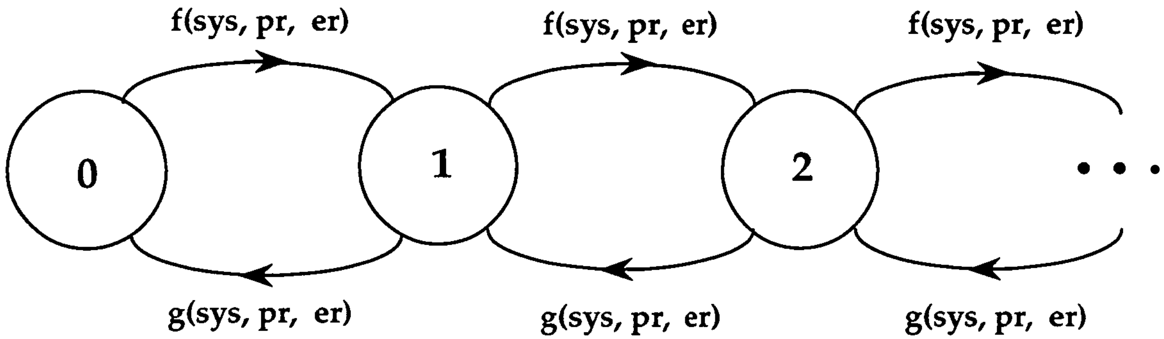
- Motivation
- Fault Content Model
- Counting Faults
- Fault surrogates
- Rate of fault injection
- Risk Assessment
- Future Work

Motivation

- Current methods of predicting software reliability don’t account for system’s structure and development process characteristics.
- To manage better a development effort, must be able to trade off between development process options, system structure options, and quality while development still in progress.
- Goals:
  - Develop improved methods of measuring a software system to assess operational risk
  - Assert better control over the system structure and the development process using these improved measures

Fault Content Model

General Model Formulations



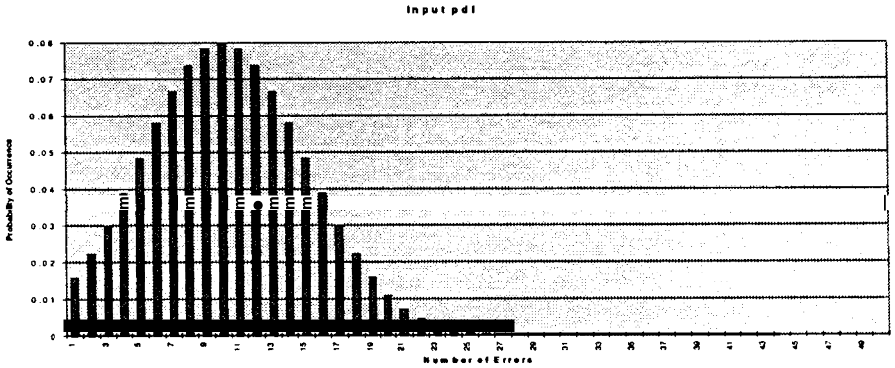
f and g are functions:

- *sys* represents characteristics of the software product
- *pr* represents characteristics of the software development process
- *er* represents the number of faults already in the system

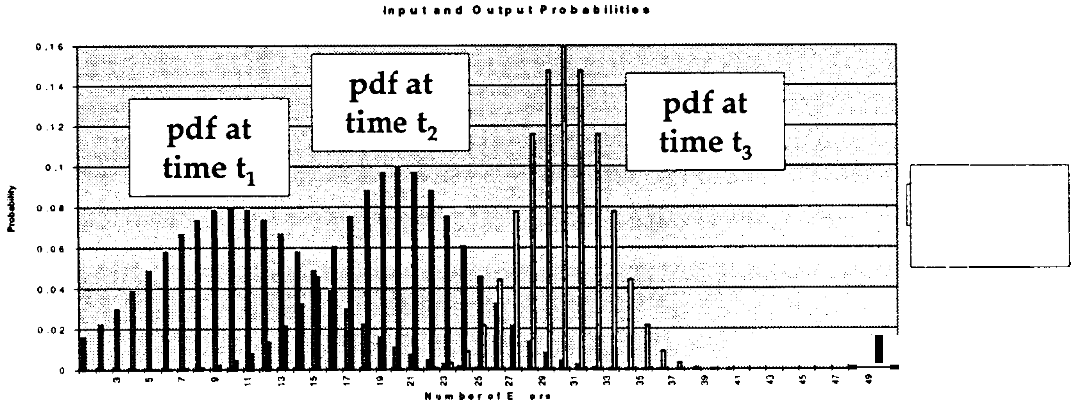
Fault Content Model (cont'd)

General Model Formulation (cont'd):

- Input:



- Output:



pdf at time t_1

pdf at time t_2

pdf at time t_3

Fault Content Model (cont'd)

Advantages of the new model:

- Ability to make resource/risk tradeoffs earlier in the development effort.
- Ability to refine and update predictions as more detailed information about product, risk, and process becomes available.
- Ability to compute confidence values.
- Predictions are in terms meaningful to users and developers.

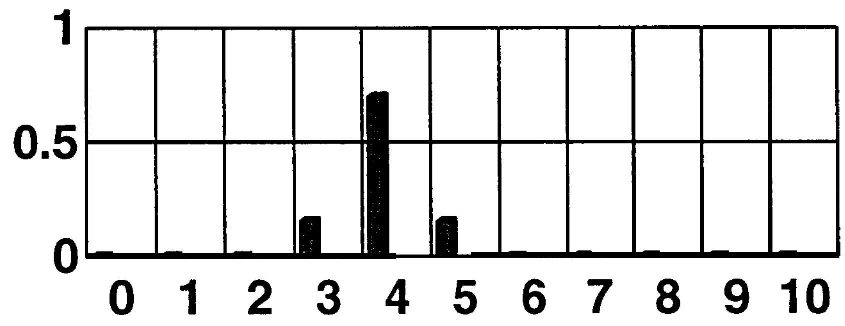
Development and use of model requires the ability to count accurately faults.

Counting Faults

- Fault vs. Failure counts
- Post-Development Fault Identification
- Fault Types
- Fault Type Composition

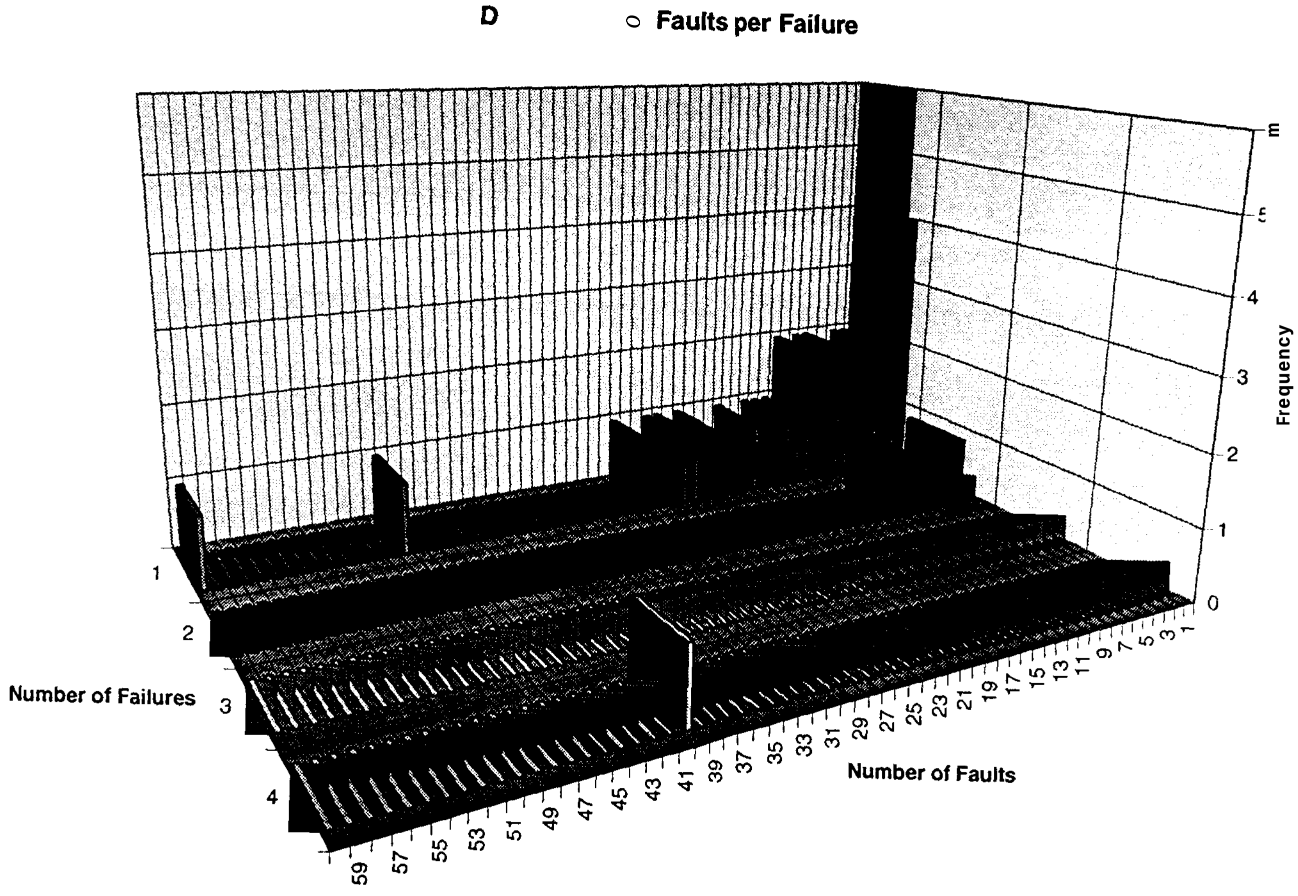
Fault vs. Failure Counts

- **Failure counts could be used if:**
  - **Number of faults related to number of failures**
  - **Distribution of number of faults per failure had low variance**



- **Actual situation is shown on next slide**

Fault vs. Failure Counts (cont'd)



Counting Faults - Post-Development Identification

- Available data
  - Institutional problem reporting systems
  - SCCS files for all delivered versions of software
- Identifying faults
  - Module repaired in increment “x” in response to a failure
  - Assume changes in increment “x” due solely to fault repair
  - Difference between “x-1” and “x” identifies changes (faults)
  - Look for earliest increments in which faults occur

Post-development fault identification is primarily a manual process

Fault Types

Taxonomy based on corrective actions taken in response to failure reports

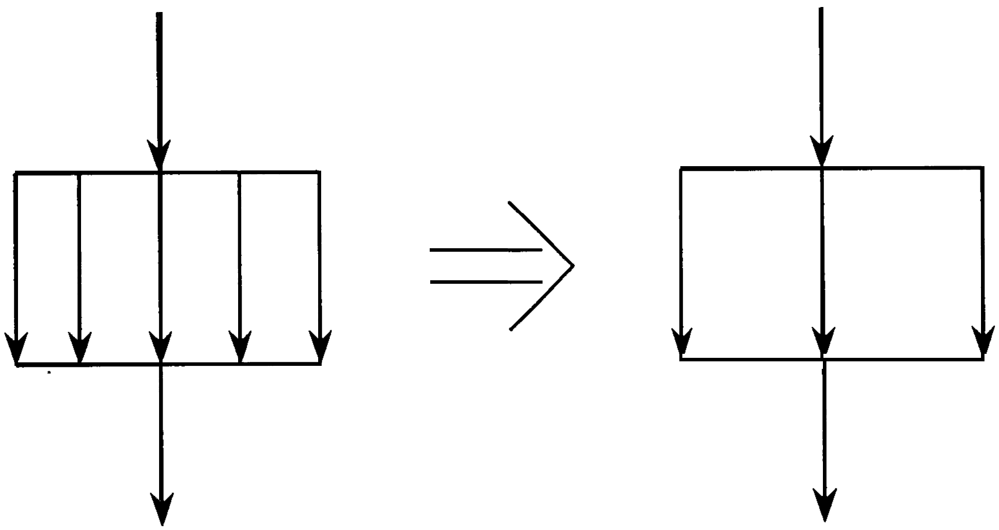
- Faults in variable usage
  - Definition and use of new variables
  - Redefinition of existing variables (e.g. changing type from float to double)
  - Variable deletion
  - Assignment of a different value to a variable
- Faults involving constants
  - Definition and use of new constants
  - Constant definition deletion

Fault Types (cont'd)

- Control flow faults
  - Addition of new source code block
  - Deletion of erroneous conditionally-executed path(s) within a source code block
  - Addition of execution paths within a source code block
  - Redefinition of condition for execution (e.g. change "if i < 9" to "if i <= 9")
  - Removal of source code block
  - Incorrect order of execution
  - Addition of a procedure or function
  - Deletion of a procedure or function

Fault Types (cont'd)

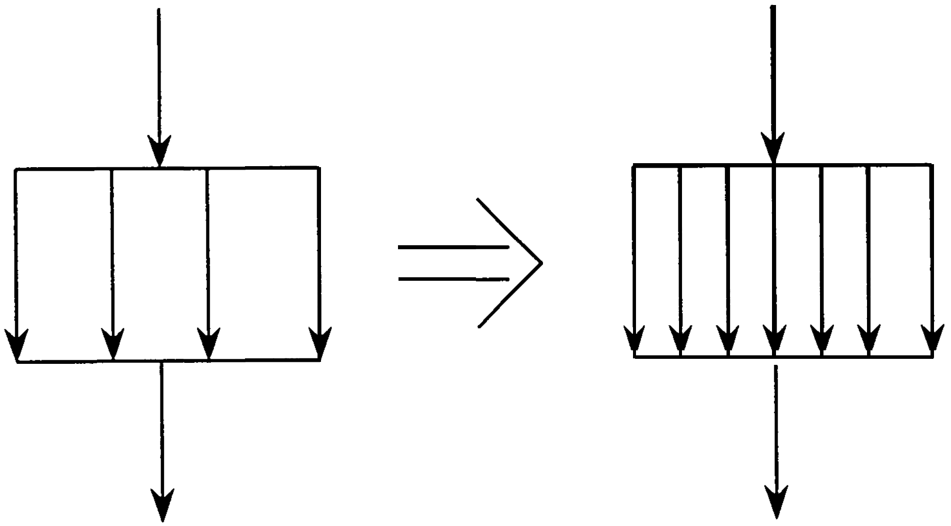
Control flow fault examples - removing execution paths from a code block



Counts as two faults, since two paths were removed

Fault Types (cont'd)

Control flow examples (cont'd) - addition of conditional execution paths to code block



Counts as three faults, since three paths were added

The Introduction of Faults

- People make errors i s the interpretation of their tasks
  - System Analysts
  - Systems Designers
  - Programmers
- These errors are manifested in
  - Specifications
  - Desigs
  - Programs

as faults

Faults and Uncertainty

- Can never know when all faults have been found
- May only use past experience to anticipate fault cou st in any reasonable manner
- We seek to develop a fault surrogate
  - Obtained estimate from past development efforts
  - Varies directly with faults
  - Anticipates distribution of faults in modules

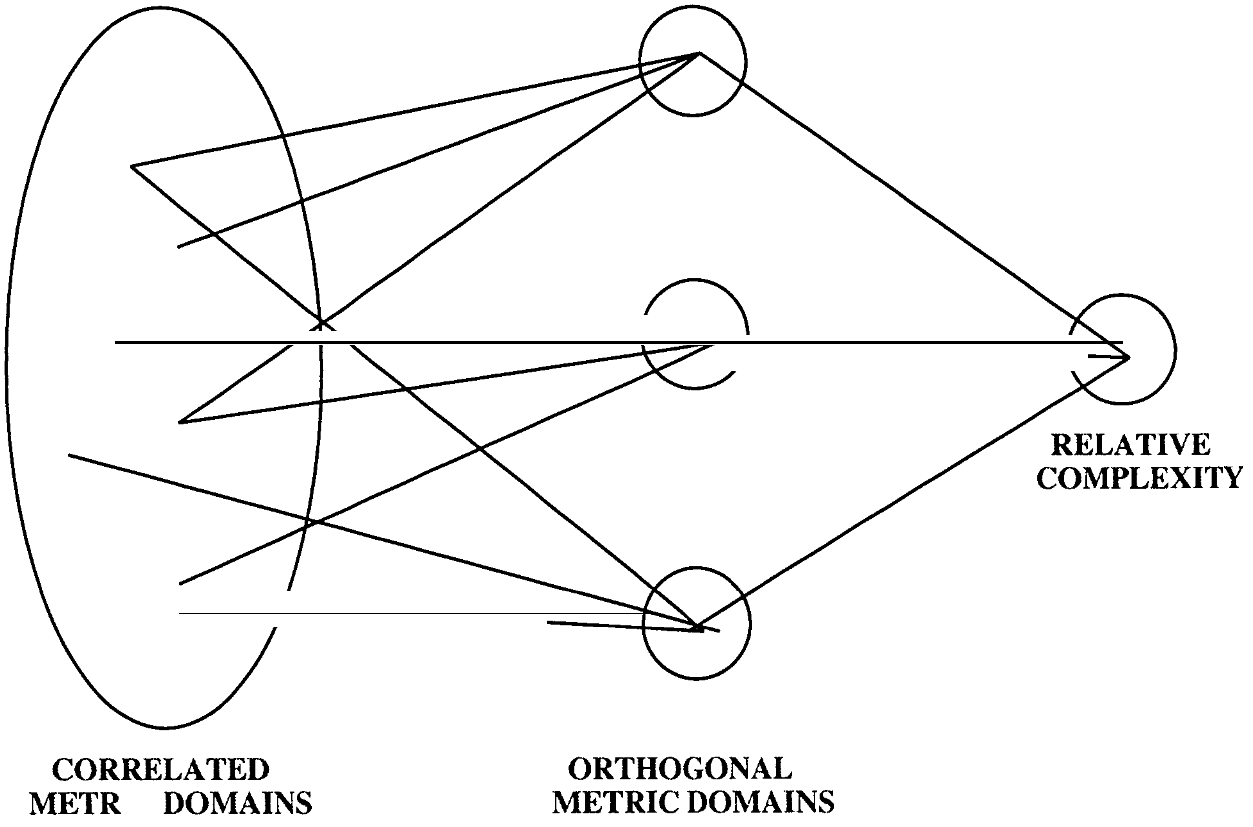
Fault Granularity

- The granularity of fault measurement must be the same as other metrics
- Changes to code are measured at the module level
- Complexity measurements are at the module level
- Configuration management is at the module level
- Faults should be maintained at the module level

Mapping of Faults

- One fault - one module
  - Fault extent within single module
- One fault - several modules
  - `include` problem
  - `COMPOOLS`
  - global data

Deriving a Fault Surrogate From Complexity Metrics



Selection of Metrics for Fault Surrogate

- Software metrics are highly correlated
- Selected for their relationship to faults
- Principal components analysis used to identify distinct sources of variation
  - The nineteen original metrics:
    - 49 295 1509 858 356 379 460 106 135
      10000 16 179 159 48 14 17 12 32
      5 54 2 45 5
  - When standardized become:
    - 3.15 1.73 0.97 0.68 2.38 1.04 1.44 1.60
      1.47 2.42 5.64 3.78 3.70 2.10 1.13 -1.10
      1.32 -0.52 1.41
  - Standardized metrics are transformed to become:
    - 3.84 0.89 0.54 -0.18

A Unitary Measure of Software Complexity

- ❖ Each complexity domain has a distinct relationship with measure of faults
- ❖ Identify complexity domains that are closely related to software faults
- ❖ Compute domain metrics for each complexity domain so related
- ❖ Relative Complexity is a weighted sum of the domain metrics

Computation of Relative Complexity

- For each program module, a s8t of measurements will be taken on selected metric primiti .8s
- Transformation coefficients t_{jk} will map the standardized raw metrics z_{ij} onto a set of domain metrics (factor scores)
- A relativ8 complexity vaШe, ρ_i will be co∩||uted for each program module as follows:

$$\rho_i = \sum_k \left(\sum_j z_{ij} t_{jk} \right) \lambda_k$$

Relative Complexity As a Fault Surrogate

- Program modules may be ordered by relative complexity
- The relative complexity of a software system is the average relative complexity of the component modules
- Relative complexity is an extensible metric
- Validation of the relative complexity concept
  - Correlates well (0.90) with measures of software faults

Relative Complexity As a Fault Surrogate

- If the relative complexity of a module is high then it will contain a large number of faults
- The metrics that comprise relative complexity were selected because they were related to faults
- If the relative complexity of a module changes during development, then the number of latent faults will also change

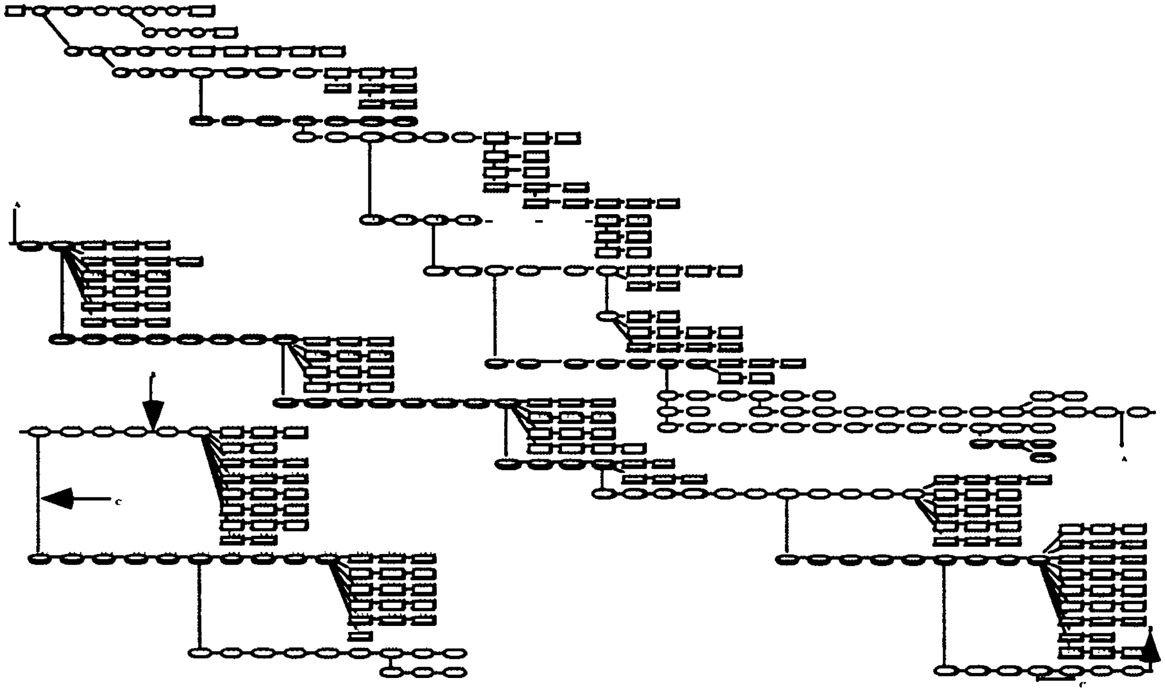
Sample Hal/S Programs Ordered by Relative Complexity

Module	Domain1	Domain2	Domain3	ρ	DR Count
1	-0.78	-0.01	0.36	43.36	0
2	-0.77	-0.02	0.36	43.37	0
3	-0.77	-0.02	0.35	43.37	0
4	-0.77	-0.02	0.34	43.39	0
5	-0.76	-0.03	0.34	43.40	0
6	-0.76	-0.00	0.31	43.53	0
7	-0.76	-0.00	0.31	43.53	0
8	3.16	3.27	2.55	95.44	9
9	7.57	-5.39	1.66	97.84	25
10	3.75	3.19	1.31	98.80	4
11	3.45	4.46	3.06	103.64	6
12	4.82	2.45	0.26	104.02	4
13	5.98	3.08	6.09	124.72	10
14	8.24	5.13	-0.86	144.42	15

Software Evolution: Measuring a Moving Target

- We assume that we are developing (maintaining) *a program*
- We are really working with many programs over time
- They are *different* programs in a very real sensCD
- We must identify and measure each version of each program modulCD

The Evolution of the Space Shuttle Pass



Managing Fault Counts During Evolution

- Some faults are inserted during branch builds
  - These fault counts must be removed when the branch is pruned
- Some faults are eliminated on branch builds
  - These faults must be removed from the main sequence build
- Fault count should contain only those faults on the main sequence to the current build
- Faults attributed to modules not in the current build must be removed from the current count

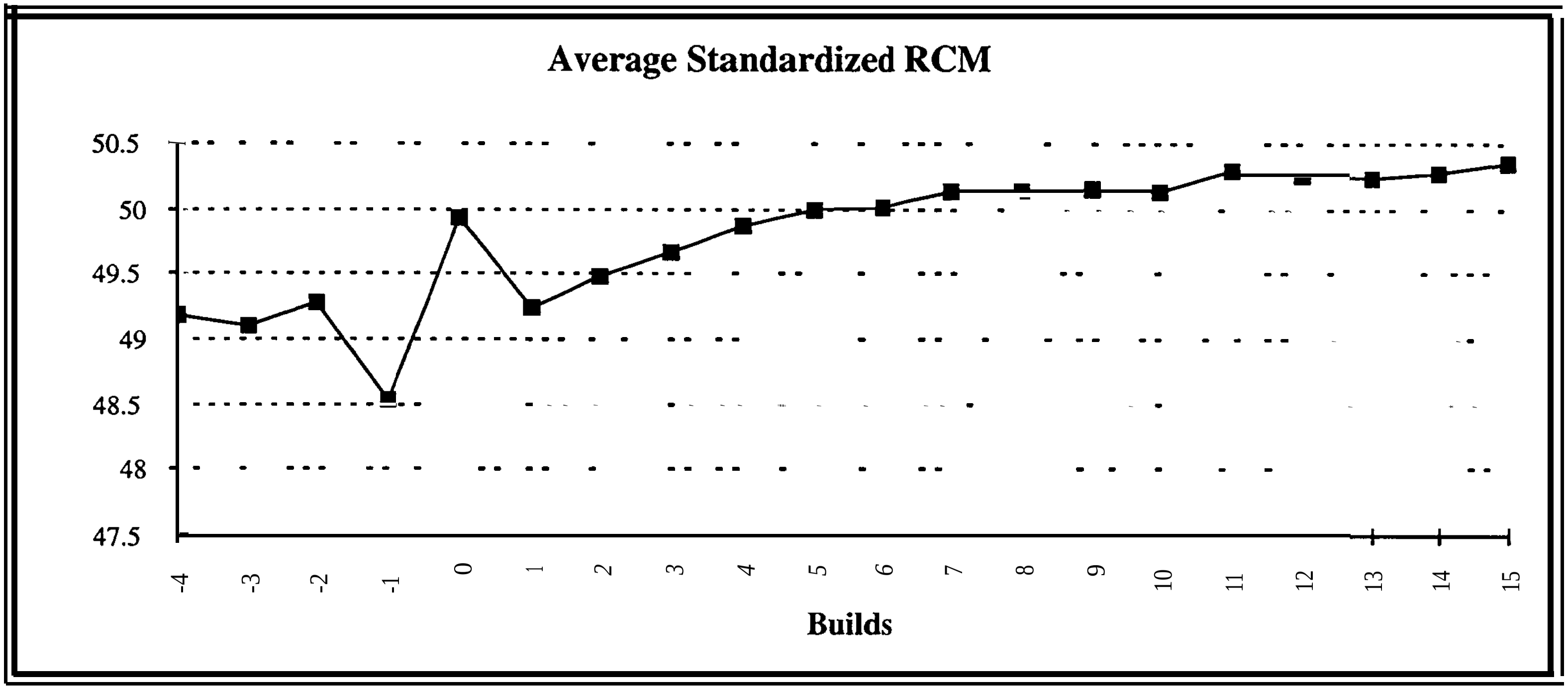
Baselining a Software Development Project

- Software changes over software builds
- Measurements, such as relative complexity, change across builds
- Initial build as a *baseline*
- Relative complexity of each build
- Measure change in fault surrogate from initial baseline

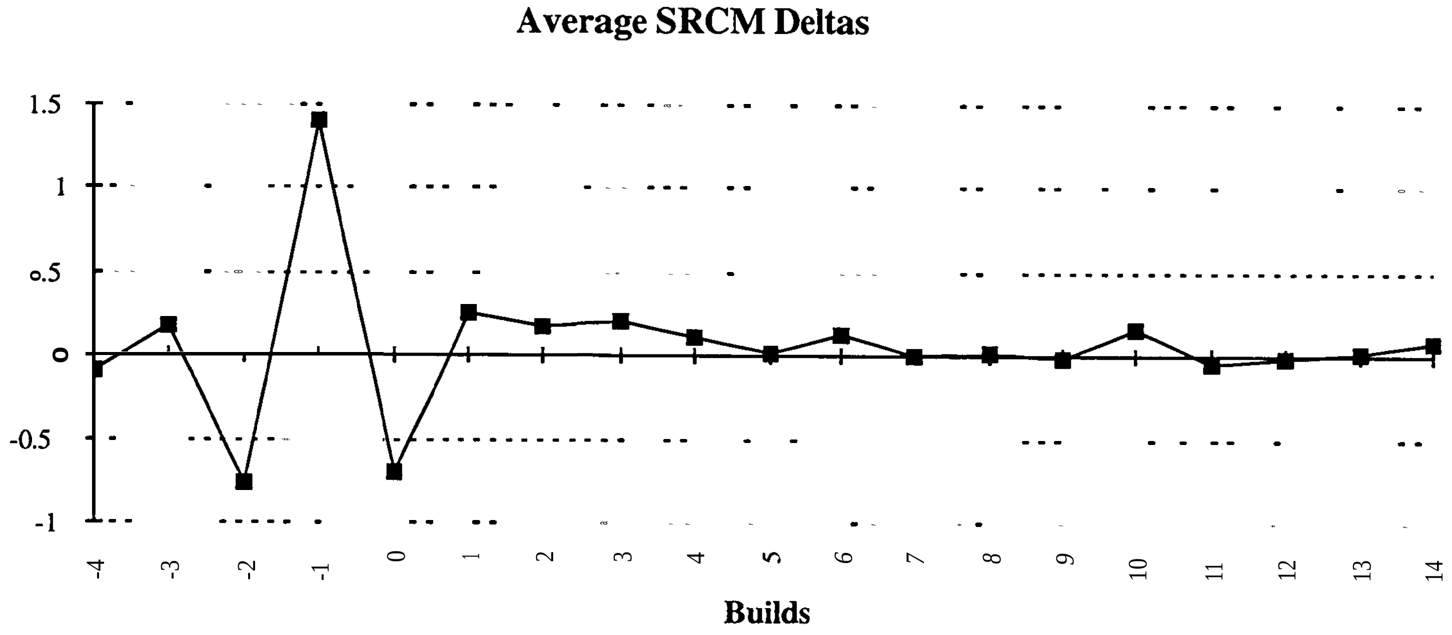
Change As a Fault Injection Process

- New faults are introduced with system changes
- Number of faults is proportional to degree of change
- Domain complexities are measures of specific changes
- Relative complexity is a measure of change
- Relative complexity is a fault surrogate

Measuring Software Evolution by Relative Complexity



Measuring Change in Fault Surrogate


Average SRCM Deltas

The Fault Injection Process During Software Development

- Immediately after the first integration test of a software system its complexity will rise in relations to the baseline complexity of the system at the first build
- The complexity of most software systems will continue to rise over most of the program's useful life
- We are continually adding functionality to existing software
- We are continually adding faults to the software in proportion to the complexity of the changes

Baselining Fault Assessment

- Total system complexity is initially $\mathbf{R} = \sum_{i=1}^{N} \rho_i$
- Initially each program module has a number of faults proportional to the fault surrogate
- Let δ_i^1 represent the proportion of faults in the i^{th} module at the first build
- The fault potential of a module *i* is proportional to its value of the relative complexity fault surrogate
- Thus,

$$r_i = \rho_i / \mathbf{R}$$

Changes to Faults

- Let L^j represent the total number of faults found at the j^{th} build of the system
- The i^{th} module will have had l_i^j faults removed
- Then $g_i^j = {l_i^j}/{L_i^j}$ represents the proportion of

  faults removed in the i^{th} module on the j^{th} build of the system
- If the changes to code to fix faults have not changed the fault surrogate measure, then the proportion of faults remaining in the i^{th} module on the j^{th} build is $\delta_i^j = \rho_i - g_i^j$

Rate of Fault Injection

- New faults will be injected into the system in proportion to the change in the fault surrogate
- The change relative complexity from build j to build $j+1$ is $\Delta_i^{j+1} = \left|\rho_i^j - \rho_i^{j+1}\right|$
- The total change over j +1 builds is

$$s_i^{j+1} = \sum_{k=2}^{j+1} \Delta_i^k$$

- New estimate for proportion of remaining faults is

$$\delta_i^{j+1} = s_i^{j+1} - g_i^{j+}$$

Execution Consequences of Faults: Failures

- A fault can only cause a failure if it is executed
- Different functionalities execute different sets of modules
- Faults can be mapped to program functionalities

Design

ASSIGNS (*f*,*m*)

$F \times M$	m_1	m_2	m_3	m_4	m_5	m_6
f_1	T	T				
f_2	T		T	T		
f_3	T		T			T
f_4	T			T		

Program Functionality

- Users specify their needs in terms of a set of operations, *O*
- Programs implemeωt the operations in a set of functionalities, *F*
- The Software Requirements Specifications define a set of relations on $O \times F$
- There is a relation IMPLEMENTS over $O \times F$
- IMPLEMENTS *(o,f)* is true if
    - functionality $f \in F$ is used to implement
    - operation $o \in O$

Specification

IMPLEMENTS (*o,f*)

$O \times F$	f_1	f_2	f_3	f_4
o_1	T	T		
o_2		T	T	T

Operation o_1 is implemented using functions f_1 and f_2

Operation o_2 is implemented using functions f_2 , f_3 and f_4

Functional Classification of Program Modules

- Some program modules will execute regardless of the functionality

  $M_c = \{m : M \mid \forall f \in F \bullet ASSIGNS \quad (f, m)\}$

- Some program modules are indispensably associated with a functionality

  $M_i^{(f)} = \{m : M_f \mid \forall f \in F \bullet \mathrm{ASSIGNS} \quad (f, m) \Rightarrow p(f, m) = 1\}$

- Some program modules may potentially execute when a given function is expressed

  $M_p^{(f)} = \{m : M_f \mid \exists f \in F \bullet ASSIGNS \quad (f, m) \wedge 0 < p(f, m) < 1\}$

Relationship of Modules to Functions

FUNCTION	M_c	M_i	M_p	M_f
f_1	$\{m_1\}$	$\{m_2, m_4\}$	$\{\ \}$	$\{m_1, m_2, m_4\}$
f_2	$\{m_1\}$	$\{m_3\}$	$\{m_5\}$	$\{m_1, m_3, m_5\}$
f_3	$\{m_1\}$	$\{\ \}$	$\{m_3, m_6\}$	$\{m_1, m_3, m_6\}$
f_4	$\{m_1\}$	$\{m_3\}$	$\{m_5, m_6\}$	$\{m_1, m_3\ m_5, m_6$

where $M_f = M_c \cup M_p^{(f)} \cup M_i^{(f}$

Operational Profile

- The Operational Profile is the set of unconditional probabilities of each of the operations being executed by a user
- Thus, $\Pr(O = o_i)$ is the probability that the user is executing an operation *i*
- The operations are mutually exclusive
- The probability distribution of the operational profile is multinomial

Functional Profile

- The Functional Profile of the software is the set of unconditional probabilities of each of the functionalities being expressed by an operation
- Thus, $\Pr(F = f_i)$ is the probability that the system is executing program functionality *i*
- The functions are mutually exclusive
- The probability distribution of the functional profile is multinomial

Execution Profile

- An Execution Profile is the conditional probability of executing a module *i* given a certain functionality *j*
- Let $Pr(M = m_j \mid F = f_i)$ represent this probability for a fixed functionality
- Underlying distribution is multinomial
- This distribution is directly determined by the program design
- We must measure to determine the distribution

Module Profile

The Module Profile is the unconditional probability that a module will be executed

$$\Pr(M_j \cap F_i) = \Pr(M_j F_i) = \Pr(F_i)\Pr(M_j \mid F_i)$$

$$\begin{aligned}\Pr(M_i) &= \sum_j \Pr(M_i F_j) \\ &= \sum_j \Pr(F_j)\Pr(M_i \mid F_j)\end{aligned}$$

Risk Assessment with Fault Surrogate as Loss Function

- At each build, an estimate for the proportion of remaining faults is $\delta_i^j = \rho_i - g_i^j$
- Each functionality has a distinct execution profile $p_\bullet^f$
- The functional risk of this execution profile is

$$\phi^f = \sum_{i=1}^{n} p_i^f \delta_i^j$$

- If a functionality is executed that will run fault prone modules with high probability, the risk (failure potential) will be high

Future Work

- ❖ Functional standards for fault recording
- ❖ Risk Assessment for software test
- ❖ New methodology for regression testing based on risk assessment
- ❖ New potential for modeling software reliability